대 한 민 국 특 허 청
KOREAN INTELLECTUAL PROPERTY OFFICE

별첨 사본은 아래 출원의 원본과 동일함을 증명함.

This is to certify that the following application annexed hereto is a true copy from the records of the Korean Intellectual Property Office.

출 원 번 호 : 10-2003-0011366
Application Number

출 원 년 월 일 : 2003년 02월 24일
Date of Application FEB 24, 2003

출 원 인 : 삼성전자주식회사
Applicant(s) SAMSUNG ELECTRONICS CO., LTD.

2003 년 06 월 13 일

특 허 청
COMMISSIONER

【서지사항】

【서류명】	특허출원서
【권리구분】	특허
【수신처】	특허청장
【제출일자】	2003.02.24
【발명의 명칭】	자바 프로그램에서 바이트 코드의 컴파일 시간 단축 시스템 및 방법
【발명의 영문명칭】	SYSTEM AND METHOD FOR SHORTENING TIME IN COMPILING OF BYTE CODE IN JAVA PROGRAM
【출원인】	
【명칭】	삼성전자 주식회사
【출원인코드】	1-1998-104271-3
【대리인】	
【성명】	김동진
【대리인코드】	9-1999-000041-4
【포괄위임등록번호】	2002-007585-8
【발명자】	
【성명의 국문표기】	송효정
【성명의 영문표기】	SONG,Hyo Jung
【주민등록번호】	691029-2041313
【우편번호】	120-091
【주소】	서울특별시 서대문구 홍제1동 367-22
【국적】	KR
【발명자】	
【성명의 국문표기】	박정규
【성명의 영문표기】	PARK,Jung Gyu
【주민등록번호】	730228-1799810
【우편번호】	130-034
【주소】	서울특별시 동대문구 답십리4동 1-424
【국적】	KR
【심사청구】	청구
【취지】	특허법 제42조의 규정에 의한 출원, 특허법 제60조의 규정에 의한 출원심사 를 청구합니다. 대리인 김동진 (인)

【수수료】

【기본출원료】	17	면	29,000 원
【가산출원료】	0	면	0 원
【우선권주장료】	0	건	0 원
【심사청구료】	13	항	525,000 원
【합계】	554,000 원		

【첨부서류】 1. 요약서·명세서(도면)_1통

【요약서】

【요약】

본 발명은 자바 프로그램 소스 코드를 컴파일하여 생성된 바이트 코드를 로딩하는 클래스 로더부와, 상기 클래스 로더부에서 로딩한 바이트 코드 및 바이트 코드를 컴파일하여 생성된 네이티브 코드를 엑세스 가능한 상태로 유지하는 제1 메모리부와, 상기 제1 메모리부에 엑세스 가능한 상태로 로딩된 네이티브 코드를 저장하는 제2 메모리부와, 상기 클래스 로더부의 요청에 따라 제2 메모리부에 저장된 네이티브 코드를 검색하여 제1 메모리부로 로딩하는 네이티브 코드 관리부 및 상기 제1 메모리부에 엑세스 상태로 로딩된 네이티브 코드를 실행시키는 실행부를 포함하는 것을 특징으로 하며, 자바 프로그램 실행시 자주 사용되는 바이트 코드를 네이티브 코드로 컴파일하여 저장해 놓음으로써, 이후에 자바 프로그램을 시행할 때 해당 바이트 코드에 대한 네이티브 코드를 불러와 실행 시킴으로써, 바이트 코드를 컴파일하여 네이티브 코드를 생성하는데 소요되는 시간을 줄일 수 있다.

【대표도】

도 1

【색인어】

자바 프로그램, 바이트 코드, 네이티브 코드

【명세서】

【발명의 명칭】

자바 프로그램에서 바이트 코드의 컴파일 시간 단축 시스템 및 방법{SYSTEM AND METHOD FOR SHORTENING TIME IN COMPILING OF BYTE CODE IN JAVA PROGRAM}

【도면의 간단한 설명】

도 1은 본 발명의 자바 프로그램에서 바이트 코드의 컴파일 시간 단축 시스템을 개략적으로 나타낸 블럭도.

도 2는 본 발명의 자바 프로그램에서 바이트 코드의 컴파일 시간 단축 방법을 개략적으로 나타낸 플로우챠트.

< 도면의 주요부분에 대한 부호의 설명 >

100 : 클래스 로더부 200 : 제1 메모리부

300 : 네이티브 코드 관리부 400 : 제2 메모리부

500 : 실행부 600 : 가비지 컬렉터부

【발명의 상세한 설명】

【발명의 목적】

【발명이 속하는 기술분야 및 그 분야의 종래기술】

<7> 본 발명은 자바 프로그램에서 바이트 코드의 컴파일 시간 단축 시스템 및 방법에 관한 것으로서, 특히 자바 프로그램 실행시 자주 사용되는 바이트 코드를 컴파일하여 생성된 네이티브 코드를 저장해 놓음으로써, 이후에 자바 프로그램을 시행할 때 해당 바이트 코드에 대한 네이티브 코드를 불러와 실행토록 하여 네이티브 코드의 컴파일 시간을

줄일 수 있는 자바 프로그램에서 바이트 코드의 컴파일 시간 단축 시스템 및 방법에 관한 것이다.

<8> 일반적으로, 자바 프로그램 소스 코드를 컴파일하여 생성된 바이트 코드(byte code)는 자바 가상 머신(Java Virtual Machine : JVM)에 의해서 실행되는데, 상기 자바 가상 머신의 실행의 속도를 향상시키기 위해서 바이트 코드를 CPU가 직접 실행할 수 있는 형태인 네이티브 코드(native code)로 바꿔서 실행하는 방법이 최근 들어 많이 이용되고 있다.

<9> 그 방법 중에서 dynamic adaptive compilation(이하, DAC라 함)은 바이트 코드를 해석(interpretation)하다가 상기 바이트 코드가 자주 사용되는(bottleneck) 바이트 코드인 경우 네이티브 코드로 컴파일(compile)하여 실행한다. 즉, 상기 DAC 방법은 전체 프로그램을 네이티브 코드로 컴파일 하지 않고 자주 사용되는 바이트 코드인 경우에만 네이티브 코드로 컴파일하는 것이다.

<10> 그러나, 바이트 코드를 실행하는 중에 만들어지는 네이티브 코드들은 메모리 부족 시 가비지 컬렉터(garbage collector)에 의해 회수되며, 또한 자바 프로그램의 실행이 끝나게 되면 모두 사라져 버려서 다음 번 실행에서 사용할 수 없다는 문제점이 있다.

<11> 그런데, 한번 네이티브 코드로 만들어지는 바이트 코드들 중 대부분은 다음 번에도 또 다시 네이티브 코드로 만들어지는 특성이 있기 때문에, 만들어진 네이티브 코드를 버리는 것은 다음 번 실행에서 또 다시 같은 네이티브 코드를 만들어야 하는 문제가 발생한다.

<12> 따라서, 기존의 방법은 무선 단말기와 같은 저 성능의 CPU와 저 용량의 배터리를 가진 시스템에서 응답 시간(response time)을 증가시키고, 배터리 소모를 많은 일으키는 문제점이 있다.

【발명이 이루고자 하는 기술적 과제】

<13> 본 발명은 상기한 문제점을 해결하기 위하여 안출된 것으로서, 본 발명의 목적은 자바 프로그램 실행시 자주 사용되는 바이트 코드를 컴파일하여 생성된 네이티브 코드를 저장해 놓음으로써, 이후에 자바 프로그램을 시행할 때 해당 바이트 코드에 대한 네이티브 코드를 불러와 실행 시킴으로써 자바 프로그램에서 바이트 코드의 컴파일 시간을 줄일 수 있는 시스템 및 방법을 제공하는 것이다.

<14> 본 발명의 다른 목적은 저 성능 CPU와 저 용량 메모리를 가진 기기에서 컴파일 시간을 줄임으로써 자바 프로그램의 실행 시간이 단축되어 사용자에 대한 응답 시간을 감소 및 배터리 소모량을 줄일 수 있는 자바 프로그램에서 바이트 코드의 컴파일 시간 단축 시스템 및 방법을 제공하는 것이다.

【발명의 구성 및 작용】

<15> 상기 목적을 달성하기 위하여 본 발명은, 자바 프로그램 소스 코드를 컴파일하여 생성된 바이트 코드를 로딩하는 클래스 로더부와, 상기 클래스 로더부에서 로딩한 바이트 코드 및 바이트 코드를 컴파일하여 생성된 네이티브 코드를 엑세스 가능한 상태로 유지하는 제1 메모리부와, 상기 제1 메모리부에 엑세스 가능한 상태로 로딩된 네이티브 코드를 저장하는 제2 메모리부와, 상기 클래스 로더부의 요청에 따라 제2 메모리부에 저장된 네이티브 코드를 검색하여 제1 메모리부로 로딩하는 네이티브 코드 관리부 및 상기

제1 메모리부에 엑세스 상태로 로딩된 네이티브 코드를 실행시키는 실행부를 포함하는 것을 특징으로 한다.

<16> 또한, 클래스 로더부가 컴파일된 바이트 코드를 로딩하는 단계와, 상기 로딩된 바이트 코드에 해당하는 네이티브 코드의 검색을 네이티브 코드 관리부에게 요청하는 단계와, 상기 요청받은 네이티브 코드를 제2 메모리부에서 검색하는 단계와, 상기 검색된 해당 네이티브 코드를 제1 메모리부에 전송하는 단계 및 상기 전송된 네이티브 코드를 네이티브 코드 실행부가 실행시키는 단계를 포함하는 것을 특징으로 한다.

<17> 이하, 첨부한 도면들을 참조로 본 발명의 바람직한 실시예를 상세히 설명한다.

<18> 도 1은 본 발명의 자바 프로그램에서 바이트 코드의 컴파일 시간 단축 시스템을 개략적으로 나타낸 블럭도로서, 클래스 로더부(class loader)(100), 제1 메모리부(200), 네이티브 코드 관리부(300), 제2 메모리부(400), 실행부(500) 및 가비지 컬렉터부(garbage collector)(600)로 구성된다.

<19> 클래스 로더부(100)는 자바 프로그램 소스 코드를 컴파일하여 생성된 바이트 코드를 로딩한다. 여기서, 상기 로딩 과정이란 보조기억장치에 위치한 바이트 코드를 자바 가상 머신으로 불러오는 것을 말한다.

<20> 제1 메모리부(200)는 상기 클래스 로더부(100)에서 로딩한 바이트 코드 및 바이트 코드를 컴파일하여 생성된 네이티브 코드를 엑세스 가능한 상태로 유지한다. 즉, 상기 클래스 로더부(100)에서 로딩한 바이트 코드 및 바이트 코드를 컴파일하여 생성된 네이티브 코드를 소정의 메모리 영역에 저장하며, 이 후 후술하는 실행부(500)가 상기 저장된 네이티브 코드를 엑세스 할 수 있도록 한다.

<21> 제2 메모리부(400)는 상기 제1 메모리부에 엑세스 가능한 상태로 로딩된 네이티브 코드를 저장한다.

<22> 네이티브 코드 관리부(300)는 상기 클래스 로더부(100)의 요청에 따라 제2 메모리부(400)에 저장된 네이티브 코드를 검색하여 제1 메모리부(200)로 로딩하며, 또한 네이티브 코드 관리부(300)는 제 1 메모리부에 로딩된 네이티브 코드를 제2 메모리부(400)에 저장한다. 여기서, 상기 네이티브 코드 관리부(300)는 LRU(Least Recently Used) 방식을 이용하여 제2 메모리부(400)에 저장된 네이티브 코드를 관리하는데, 상기 LRU 방식이란 저장된 네이티브 코드들 중 자주 사용되지 않는 네이티브 코드를 체크하여 잘 사용되지 않는 순으로 해당 데이터를 버리는 것으로, 여기서 상기 자주 사용되지 않는 네이티브 코드를 버리는 기준은 예를 들어 메모리 용량 또는 저장된 시간에 따라 결정될 수 있다.

<23> 실행부(500)는 상기 제1 메모리부(200)에 엑세스 상태로 로딩된 네이티브 코드 및 바이트 코드를 실행시키는 것으로, 바이트 코드 인터프리터(byte code interpreter)(510), 런타임 프로파일러(runtime profiler)(520), 네이티브 코드 컴파일러(native code compiler)(530), 네이티브 코드 실행부(native code execution)(540)를 포함한다.

<24> 바이트 코드 인터프리터(510)는 상기 제 1 메모리부(200)에 엑세스 상태로 로딩된 바이트 코드를 해석하여 실행시킨다.

<25> 런타임 프로파일러(520)는 상기 바이트 코드 인터프리터(510)에서 해석되는 바이트 코드가 자주 사용되는 바이트 코드인지를 체크하고, 상기 체크 결과를 네이티브 코드 컴파일러(530)에 알려준다.

<26> 네이티브 코드 컴파일러(530)는 상기 런타임 프로파일러(520)의 체크 결과 상기 바이트 코드 인터프리터(510)에서 해석중인 바이트 코드가 자주 사용되는 바이트 코드인 경우, 상기 해석중인 바이트 코드를 컴파일하여 네이티브 코드를 생성한다. 또한 상기 네이티브 코드 컴파일러(530)는 상기 컴파일된 네이티브 코드를 제1 메모리부(200)로 로딩한다.

<27> 네이티브 코드 실행부(540)는 상기 네이티브 코드 관리부(300)가 제1 메모리부에 로딩한 네이티브 코드를 실행 시킨다.

<28> 가비지 컬렉터부(600)는 상기 제1 메모리부(200)에서 더이상 실행되지 않는 코드가 차지하는 공간들을 자동으로 회수하는 것으로, 이로 인해 상기 제1 메모리부(200)의 공간을 확보해 준다. 또한, 상기 가비지 컬렉터부(600)가 더이상 실행되지 않는 코드들을 회수한 후에도 제1 메모리부(200)의 공간이 부족할 경우, 네이티브 코드 관리부(300)에게 네이티브 코드를 제2 메모리부(400)에 저장하도록 요청한다.

<29> 도 2는 본 발명의 자바 프로그램에서 바이트 코드의 컴파일 시간 단축 방법을 개략적으로 나타낸 플로우챠트이다.

<30> 먼저, 클래스 로더부(100)가 자바 프로그램 소스 코드를 컴파일하여 생성된 바이트 코드를 로딩한 후(S100), 네이티브 코드 관리부(300)에게 상기 로딩된 바이트 코드에 해당하는 네이티브 코드의 검색을 요청하면(S110), 상기 네이티브 코드 관리부(300)는 요청받은 네이티브 코드를 제2 메모리부(400)에서 검색한다(S120).

<31> 상기 제2 메모리부(400)에서 해당 네이티브 코드가 검색되면(S130), 상기 검색된 네이티브 코드를 제1 메모리부(200)에 전송하고(S132), 네이티브 코드 실행부(540)는 상

기 제1 메모리부(200)에 전송된 네이티브 코드를 실행시킨다(S134). 여기서, 상기 제2 메모리부(400)에 저장된 네이티브 코드는 자바 프로그램 실행시 자주 사용되는 바이트 코드를 컴파일하여 생성된 네이티브 코드이다.

<32> 본원 발명에서는 제2 메모리부(400)에 저장된 네이티브 코드를 제1 메모리부(200)에 로딩하여 실행만 시키면 되므로, 바이트 코드를 매번 해석하여 실행하지 않아도 되며, 이로써 바이트 코드의 인터프리테이션(interpretation) 과정 수행 과정을 생략할 수 있다.

<33> 한편, 상기 네이티브 코드 관리부(300)가 제2 메모리부(400)를 검색한 결과 해당 바이트 코드에 대한 네이티브 코드가 존재하지 않은 경우, 상기 클래스 로더부(100)는 로딩된 바이트 코드를 제1 메모리부(200)로 전송한다(S140).

<34> 상기 제1 메모리부(200)에 바이트 코드가 로딩되면, 바이트 코드 인터프리터(510)는 상기 로딩된 바이트 코드가 실행될 수 있도록 해석(interpretation)한다(S150).

<35> 상기 바이트 코드 인터프리터(510)가 바이트 코드 해석시, 런타임 프로파일러(520)는 상기 바이트 코드 인터프리터(510)에서 해석되는 바이트 코드가 자주 사용되는 바이트 코드인지를 체크하고(S160), 그 결과를 네이티브 코드 컴파일러(530)로 전송한다.

<36> 상기 체크 결과 자주 사용되는 바이트 코드인 경우(S170), 상기 해석된 바이트 코드는 네이티브 코드 컴파일러(530)로 전송되어 컴파일됨으로써 네이티브 코드를 생성한다(S172). 여기서, 상기 네이티브 코드는 제1 메모리부(200)로 전송되고, 이 후 네이티브 코드 관리부(300)에 의해 제2 메모리부(400)에 저장된다(S174). 여기서, 상기 제2 메모리부(400)에 저장된 네이티브 코드는 네이티브 코드 관리부(300)에 의해 LRU 방식으로

관리된다. 즉, 제2 메모리부(400)의 저장 영역이 한정되어 있기 때문에 LUR 방식을 적용하여 저장된 네이티브 코드를 관리하는 것이다.

<37> 한편, 상기 체크 결과, 자주 사용되지 않는 바이트 코드이거나, 처음 사용되는 바이트 코드인 경우, 상기 바이트 코드 인터프리터(510)에서 해석되어 실행된다(S180).

<38> 또한, 가비지 컬렉터부(600)는 상기 제1 메모리부(200)에서 더이상 실행되지 않는 코드가 차지하는 공간들을 자동으로 회수함으로써, 상기 제1 메모리부(200)의 공간을 확보해 주며, 만일 더이상 실행되지 않는 코드들을 처리한 후에도 제1 메모리부(200)의 공간이 부족할 경우, 네이티브 코드 관리부(300)에게 제1 메모리부에 로딩되어 있는 네이티브 코드를 제2 메모리부(400)에 저장하도록 요청함으로써 제1 메모리부(200)의 공간을 확보한다.

<39> 이상에서 본 발명에 대하여 상세히 기술하였지만, 본 발명이 속하는 기술 분야에 있어서 통상의 지식을 가진 사람이라면, 첨부된 청구범위에 정의된 본 발명의 정신 및 범위를 벗어나지 않으면서 본 발명을 여러 가지로 변형 또는 변경하여 실시할 수 있음은 자명하며, 따라서 본 발명의 실시예에 따른 단순한 변경은 본 발명의 기술을 벗어날 수 없을 것이다.

【발명의 효과】

<40> 상기한 구성의 본 발명에 의하면, 자바 프로그램 실행시 자주 사용되는 바이트 코드를 컴파일하여 생성된 네이티브 코드를 저장해 놓음으로써, 이후에 자바 프로그램을 시행할 때 해당 바이트 코드에 대한 네이티브 코드를 불러와 실행 시킴으로써 바이트 코

드를 컴파일하여 네이티브 코드를 생성하는데 소요되는 시간을 줄일 수 있는 잇점이 있다.

<41> 또한, 핸드폰과 같은 저 성능 CPU와 저 용량 메모리를 가진 기기에서 컴파일 시간을 줄임으로써 자바 프로그램의 실행 시간이 단축되어 사용자에 대한 응답 시간을 감소 및 배터리 소모량을 줄일 수 있는 잇점이 있다.

【특허청구범위】

【청구항 1】

자바 프로그램 소스 코드를 컴파일하여 생성된 바이트 코드를 로딩하는 클래스 로더부;

상기 클래스 로더부에서 로딩한 바이트 코드 및 바이트 코드를 컴파일하여 생성된 네이티브 코드를 엑세스 가능한 상태로 유지하는 제1 메모리부;

상기 제1 메모리부에 엑세스 가능한 상태로 로딩된 네이티브 코드를 저장하는 제2 메모리부;

상기 클래스 로더부의 요청에 따라 제2 메모리부에 저장된 네이티브 코드를 검색하여 제1 메모리부로 로딩하는 네이티브 코드 관리부; 및

상기 제1 메모리부에 엑세스 상태로 로딩된 네이티브 코드를 실행시키는 실행부를 포함하는 것을 특징으로 하는 자바 프로그램에서 바이트 코드의 컴파일 시간 단축 시스템.

【청구항 2】

제 1항에 있어서,

상기 제1 메모리부에서 더이상 실행되지 않는 코드가 차지하는 공간들을 자동으로 회수하는 가비지 컬렉터부를 더 포함하는 것을 특징으로 하는 자바 프로그램에서 바이트 코드의 컴파일 시간 단축 시스템.

【청구항 3】

제 2항에 있어서,

상기 가비지 컬렉터부는 제1 메모리부의 공간이 부족할 경우, 네이티브 코드 관리부에게 제 1 메모리부에 로딩된 네이티브 코드를 제2 메모리부에 저장하도록 요청하는 것을 특징으로 하는 자바 프로그램에서 바이트 코드의 컴파일 시간 단축 시스템.

【청구항 4】

제 1항에 있어서, 상기 네이티브 코드 관리부는,

상기 제 1 메모리부에 로딩된 네이티브 코드를 제2 메모리부에 저장하는 것을 특징으로 하는 자바 프로그램에서 바이트 코드의 컴파일 시간 단축 시스템.

【청구항 5】

제 1항 또는 4항에 있어서, 상기 네이티브 코드 관리부는,

LRU 방식을 이용하여 상기 제2 메모리부에 저장된 네이티브 코드를 관리하는 것을 특징으로 하는 자바 프로그램에서 바이트 코드의 컴파일 시간 단축 시스템.

【청구항 6】

제 1항에 있어서, 상기 실행부는,

상기 제 1 메모리부에 엑세스 상태로 로딩된 바이트 코드를 해석하여 실행시키는 바이트 코드 인터프리터;

상기 바이트 코드 인터프리터에서 해석되는 바이트 코드가 자주 사용되는 바이트 코드인지를 체크하는 런타임 프로파일러; 및

상기 런타임 프로파일러에서 체크한 결과 자주 사용되는 바이트 코드인 경우, 네이티브 코드로 변경 하는 네이티브 코드 컴파일러를 더 포함하는 것을 특징으로 하는 자바 프로그램에서 바이트 코드의 컴파일 시간 단축 시스템.

【청구항 7】

클래스 로더부가 컴파일된 바이트 코드를 로딩하는 단계;

상기 로딩된 바이트 코드에 해당하는 네이티브 코드의 검색을 네이티브 코드 관리부에게 요청하는 단계;

상기 요청받은 네이티브 코드를 제2 메모리부에서 검색하는 단계;

상기 검색된 해당 네이티브 코드를 제1 메모리부에 전송하는 단계; 및

상기 전송된 네이티브 코드를 네이티브 코드 실행부가 실행시키는 단계를 포함하는 것을 특징으로 하는 자바 프로그램에서 바이트 코드의 컴파일 시간 단축 방법.

【청구항 8】

제 7항에 있어서, 상기 제2 메모리부에 저장된 네이티브 코드는,

네이티브 코드 관리부에 의해 LRU 방식을 통해 관리되는 것을 특징으로 자바 프로그램에서 바이트 코드의 컴파일 시간 단축 방법.

【청구항 9】

제 7항에 있어서, 상기 제2 메모리부에서 네이티브 코드 검색 결과 해당 네이트브 코드가 존재하지 않을 경우에는,

클래스 로더부가 로딩한 바이트 코드를 제1 메모리부로 전송하는 단계;

상기 제1 메모리부로 전송된 바이트 코드를 바이트 코드 인터프리터에서 해석하여 실행시키는 단계를 더 포함하는 것을 특징으로 하는 자바 프로그램에서 바이트 코드의 컴파일 시간 단축 방법.

【청구항 10】

제 9항에 있어서, 제1 메모리부로 전송된 바이트 코드를 바이트 코드 인터프리터에서 해석하여 실행시키는 단계는,

런타임 프로파일러가 상기 바이트 코드 인터프리터에서 해석되는 바이트 코드가 자주 사용되는 바이트 코드인지를 체크하는 단계를 더 포함하는 것을 특징으로 하는 자바 프로그램에서 바이트 코드의 컴파일 시간 단축 방법.

【청구항 11】

제 10항에 있어서, 상기 체크 결과 자주 사용되는 바이트 코드인 경우,

네이티브 코드 컴파일러가 바이트 코드 인터프리터에서 해석된 바이트 코드를 컴파일하여 네이티브 코드를 생성하는 단계;

상기 생성된 네이티브 코드를 제1 메모리부에 로딩하는 단계; 및

상기 로딩된 네이티브 코드를 네이티브 코드 관리부가 제2 메모리부에 저장하는 단계를 더 포함하는 것을 특징으로 하는 자바 프로그램에서 바이트 코드의 컴파일 시간 단축 방법.

【청구항 12】

제 9항에 있어서, 상기 제1 메모리부에 로딩된 네이티브 코드는,

자바 프로그램의 실행 종료시 또는 제1 메모리부 공간 부족시 제2 메모리부에 저장되는 것을 특징으로 하는 자바 프로그램에서 바이트 코드의 컴파일 시간 단축 방법.

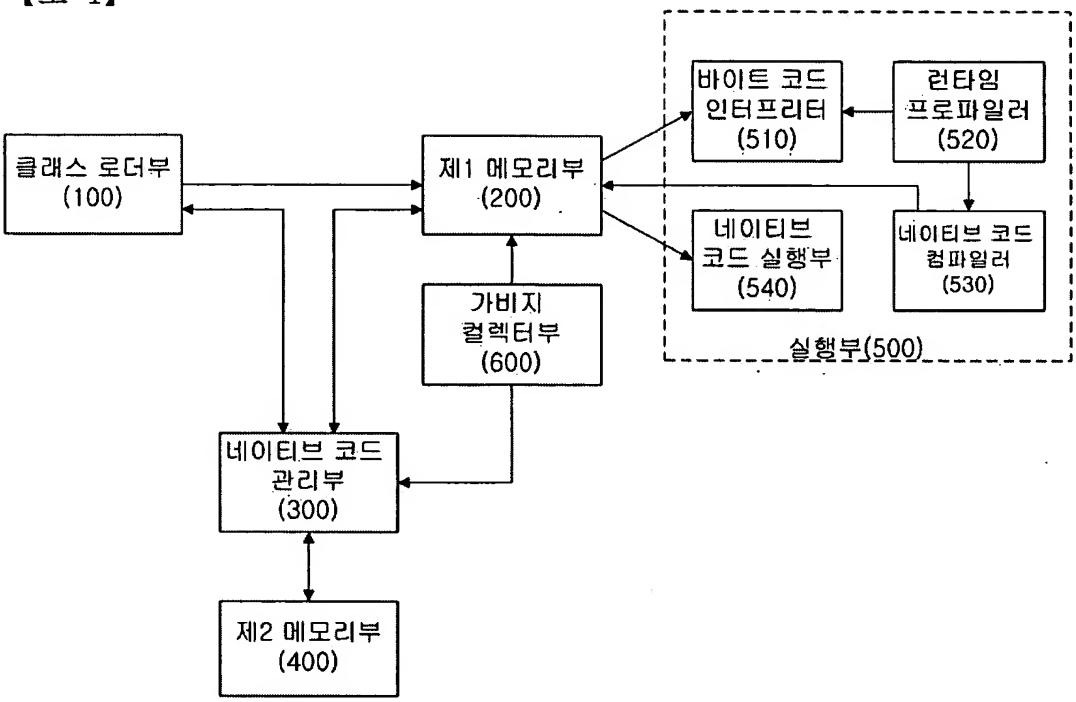
【청구항 13】

제 9항 또는 12항에 있어서, 상기 제2 메모리부에 저장된 네이티브 코드는,

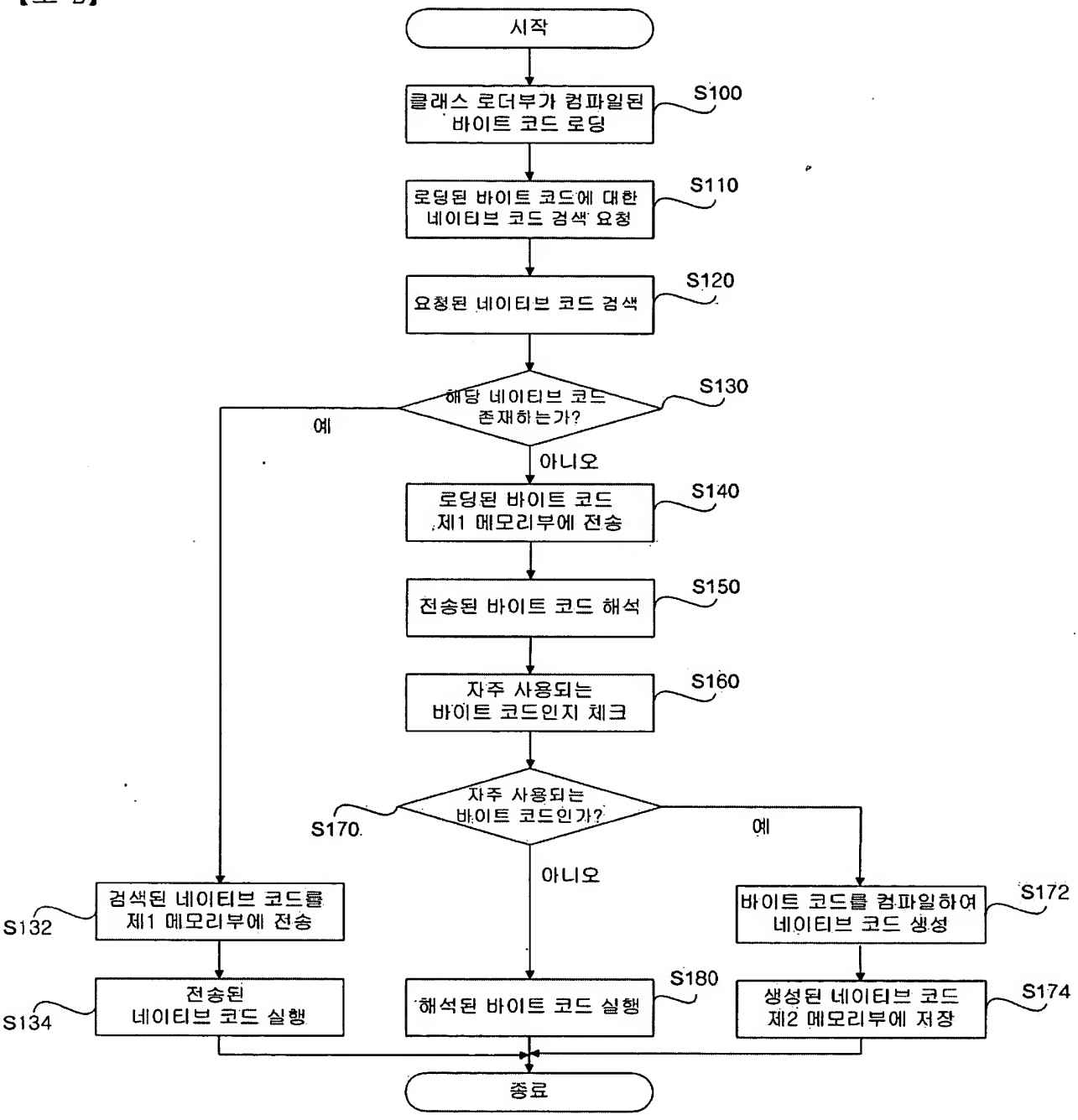
네이티브 코드 관리부에 의해 LRU 방식을 통해 관리되는 것을 특징으로 하는 자바 프로그램에서 바이트 코드의 컴파일 시간 단축 방법.

【도면】

【도 1】


클래스 로더부
(100)
제1 메모리부
(200)
바이트 코드
인터프리터
(510)
런타임
프로파일러
(520)
네이티브
코드 실행부
(540)
네이티브 코드
컴파일러
(530)
실행부(500)
가비지
컬렉터부
(600)
네이티브 코드
관리부
(300)
제2 메모리부
(400)

【도 2】


시작
클래스 로더부가 컴파일된 바이트 코드 로딩
S100
로딩된 바이트 코드에 대한 네이티브 코드 검색 요청
S110
요청된 네이티브 코드 검색
S120
해당 네이티브 코드 존재하는가?
S130
예
아니오
로딩된 바이트 코드 제1 메모리부에 전송
S140
전송된 바이트 코드 해석
S150
자주 사용되는 바이트 코드인지 체크
S160
자주 사용되는 바이트 코드인가?
S170
예
아니오
검색된 네이티브 코드를 제1 메모리부에 전송
S132
바이트 코드를 컴파일하여 네이티브 코드 생성
S172
전송된 네이티브 코드 실행
S134
해석된 바이트 코드 실행
S180
생성된 네이티브 코드 제2 메모리부에 저장
S174
종료